Pre-print version
D. Ardagna, C. Francalanci, V. Piuri, F. Scotti, "Evolutionary Design of Information Systems Architectures", in Proc. of the 2004 International Conference on Artificial Intelligence and Soft Computing (ICAISC 2004), Zakopane, Poland, pp. 1-8, June, 2004 [DOI: 10.1007/978-3-540-24844-6_1]

Evolutionary Design of Information Systems Architectures

Danilo Ardagna[1], Chiara Francalanci[1], Vincenzo Piuri[2], and Fabio Scotti[2]

[1] Politecnico di Milano,Department of Electronics and Information
piazza L. da Vinci 32, 20133 Milano (Mi), Italy
{ardagna,francala}@elet.polimi.it,
[2] University of Milan Department of Information Technologies,
via Bramante 65, 26013 Crema (Cr), Italy
{fscotti,piuri}@dti.unimi.it

Abstract. Information system design and optimum sizing is a very complex task. Theoretical research and practitioners often tackle the optimization problem by applying specific techniques for the optimization of individual design phases, usually leading to local optima. Conversely, this paper proposes the definition of a design methodology based on an evolutionary approach to the optimization of the client/server-farm distributed structure, which is typical of a distributed information technology (IT) architecture. The optimization problem consists of finding the minimum-cost physical systems that satisfy all architectural requirements given by the designer. The proposed methodology allows for the identification of the architectural solution that minimizes costs, against different information system requirements and multiple design alternatives, thorough a genetic-based exploration of the solution space. Experimental results show that costs can be significantly reduced with respect to conventional approaches adopted by IT designers and available in the professional literature.

1 Introduction

Information system design and sizing constitute a complex, top-down process that tailors the technology architecture to application requirements. Practitioners usually tackle this top-down process by focusing on individual design aspects, such as data or specific applications, and by relying on their previous experiences to compare alternative architectural solutions. Acquisition costs are usually accounted for, but related operating and maintenance costs are often neglected or underestimated. The complexity of optimizing individual design problems leads researchers to avoid a global optimization perspective and, thus, the IT architecture is usually a result of the juxtaposition of multiple local-optima. The historical cost-minimizing design principle was centralization, which was advocated to take advantage of hardware scale economies according to Grosch's law. A further attempt to formulate a general design principle has been made in the mid '80s, when Grosch's law has been revised as "It is most cost effective to accomplish any task on the least powerful type of computer capable of performing

it" [3]. Decentralization and its operating rule, referred to as downsizing, became the methodological imperative for cost-oriented infrastructural design. Although academic studies have challenged the generality of the decentralization principle ([4], [5]), the empirical recognition of the cost disadvantages of decentralization has only occurred in the '90s with the observation of client-server costs. Addressing this failure, empirical studies have showed that initial acquisition expenses represent at most 20% of the total cost of a computer over its life cycle and as a consequence, the minimization of acquisition costs does not deliver the most convenient infrastructural design solutions. The concept of "total cost of ownership" (TCO) has been introduced and defined as the summation of both investments and management costs of infrastructural components. It has been observed that while decentralization reduces investment costs, it increases management costs, due to a more cumbersome administration of a greater number of infrastructural components. Recentralization has been thereafter considered to reduce management costs and thin clients (TCs) are currently proposed as a less expensive alternative to personal computers that can be exploited through a recentralization initiative [8]. Thin clients have lower computing capacity than PCs, which is sufficient for the execution or the emulation of the presentation tier, but requires the recentralization of the application logic on a server and have management costs 20-35% lower than personal computers ([8],[9]). Furthermore, the Independent Computing Architecture (ICA) and Remote Desktop Protocol (RDP) standards allow remote access to the application logic by traditional PCs. This translates into hybrid configurations of PCs which execute only a subset of applications which will be referred to in the following as hybrid fat clients (HFCs) (see also Section 2). Thin client can be supported by multiple coordinated machines, known as server farm [7]. This load sharing allows downsizing and reduces acquisition costs. Furthermore, it has limited negative effects on management costs, since server farms are equipped with software tools that allow the remote management and simultaneous administration of all servers [10]. In previous works ([1], [2]), the problem of design of IT architecture has been represented as a single cost minimization problem. Optimization was accomplished by implementing a tabu-search algortihm. In this paper we consider the problem of design of the client architecture and the optimization of the server farm which supports thin client and HFC systems by implementing a genetic algorithm. A comparison of the two methods allow to verify the quality of the solution that can be obtained by a heuristic approach.

2 Client Technologies in Modern IT Architectures

In the '90s, two-tier client/server systems were classified by Gartner Group as Distributed Presentation, Remote Presentation, etc. [1]. From the hardware point of view, client systems can be classified as fat, i.e. traditional PCs, and thin. A thin client is a computing device that enables users to remotely perform computing tasks on a server. A thin client is diskless and cannot work without being connected to a server; the physical desktop unit only needs enough

memory and computing power to recognize keystrokes and mouse events and to display pixel data received from the server. No computing is performed locally. Considering this latter characteristic, a thin client is very similar to the dumb terminal of host-based systems. Nowadays, thin clients allow access to graphical applications and traditional PC applications. The main advantage of thin-client architectures is the reduction of TCO. The physical device is virtually free of mechanical breakdowns and the absence of local files and applications means that the desktop does not have to be administered; users are administered centrally on the server. This implies that thin clients have management costs 20-35% lower than personal computers ([8], [9], [10], [11]). Furthermore, the ICA and RDP standards allow remote access to the application logic by traditional PCs as well as new hand-held devices. This translates into hybrid fat clients which execute only a subset of client and monolithic applications. Hybrid configuration are considered since thin client architecture are not suited to support the execution of CPU intensive applications while the TCO of applications executed remotely at the server and accessed by HFCs can be reduced by 20-35% [8]. 75% of the thin clients' market is constituted by Windows Based Terminals (WBT). WBTs are the result of a joint project between Microsoft and Citrix [10]. The computational model is based on two main components: a multi-user operating system and a remote presentation protocol (RDP or ICA). Nowadays the multi-user operating system adopted is Microsoft Windows 2000/2003 (the first implementation was on Windows NT in the late '90s). The operating system has to support the execution of multiple user sessions and thin/HFC clients access to the server. The remote presentation protocol implements the message delivery of thin client inputs and desktop devices screen updates. The remaining 25% of the thin-client market is shared among X-terminals, Linux terminals and Sun Ray systems. The latter is an evolution of the Network Computer project where the desktop device is also able to locally execute the browser and some Java applications. In this work, only WBT devices will be considered, since more widespread and suitable for the optimization of the IT architecture.

3 Optimization of the Client Architecture

In [1] and [2] a general framework for the design of IT architectures has been presented. User classes and applications used by user classes are the main requirement variables which define the solution domain for the IT architecture. Users are divided in user classes C_i characterized by the same set of applications. The percentage of concurrent users is also specified. Each user class can be assigned to multiple types of client architectures (thin, fat or hybrid). User classes assigned to thins or HFCs can also share the server farm in order to reduce cost. This is obtained formally by defining groups. Note that, if a group includes n user classes, then $2^n - 1$ configurations have to be analyzed (that is the power set of the group excluding the empty set). Furthermore, for HFCs some applications are assigned to the fat PC (for example if an application is CPU intensive) or to the server (for example a data entry client application)

but some applications can be indifferently executed by the fat PC or by the server exploiting the cost trade-off that a remote execution implies. The remote execution reduces management costs while increases the computing capacity of the remote server and hence investment costs. If a class of users uses a set o n applications that can be executed indifferently by the HFC or by the remote server then $2^n - 1$ configurations have to be evaluated. The computing capacity of PCs is obtained as the maximum value of MIPS required by locally executed applications.

Similarly, computing capacity of thin/HFC servers (servers supporting thin and HFC clients) is evaluated as the maximum value of MIPS required by applications that are executed remotely, considering the number of concurrent users of the corresponding user class ([5], [1], [2]). RAM requirements are evaluated in the same way by considering the total memory required by applications. The problem of the design of the optimum configuration of server farms for thin/HFC users can be modeled as a set-partitioning problem ([2], [12]). Table 1 shows how the overall cost of the system can be evaluated. The system can be partitioned in two components: thin/HFC servers and clients. The cost of the two components can be evaluated by considering software costs, hardware costs and maintenance costs. Hardware costs include investment costs and installation costs ($Cost_{ServerFarmHW}$, $Cost_{SingleClientHW}$). Software costs include license and installation costs. Maintenance costs for hardware and software are calculated over a period of three years.

The genetic algorithm represents the system to be optimized as a chromosome. Different values of the genes of the chromosome represent different configurations of the system (solutions). The basic idea consists of maintaining and/or manipulating a family, or population, of solutions based on the so called *survival of the fittest* strategy in searching the best solution. It also gives the possibility to explore some sub-domains of the solution space at the same time [13] achieving an intrinsic parallelism.

Table 1. Evaluation of total system cost.

$$
\begin{aligned}
Cost_{System} &= Cost_{Thin/HFCServers} + Cost_{Clients} \\
Cost_{Thin/HFCServers} &= ServerHWCost(MIPS, RAM) \\
Cost_{Clients} &= \sum_{j \in Classes} NumberOfUser(j) Cost_{SingleClient}(j) \\
Cost_{SingleClient}(j) &= Cost_{SingleClientHW} + Cost_{SingleClientSW} \\
Cost_{SingleClientHW}(j) &= ClientHWCost(MIPS, RAM) \\
Cost_{SingleClientSW}(j) &= \sum_{i \in Application(j)} (Cost_{SW}(i, Location)) + (Cost_{Maint}(i, Location))
\end{aligned}
$$

Location = 1 : Application i is resident on the server
Location = 0 : Application i is resident on the client

A genetically-optimized solution to the presented problem requires the definition of the following issues: the representation of the solution as a chromosome,

the creation of the selection function for the population of the current solutions, the genetic operators used by the optimization process, the initialization, the termination and the function that drives the optimization. Figure 1 represents how it is possible to map the configuration of the clients-server system into a chromosome. In the chromosome (the integer array at the bottom of the figure) a set of integers is assigned to each user class. In each set,the first integer corresponds to the server farm SF_i that the user class has been allocated. This integer can typically range from 0 to the number of user classes. Zero means a class made of fat clients. The opposite situation is encountered when each group of applications has its server farm. Subsequent integers represent the locations of the applications. The applications have only two states: 1 if application is allocated on the server farm and 0 if it is allocated on the client. Hence, one integer is present for each application in the current user class. Each system configuration can be directly mapped into the chromosome by proper mutation of the integers.

Figure 2 shows the application of mutation and cross-over operators to the chromosome plotted in Figure 1. All applications of class C_2 are now allocated on a server (integers corresponding to A_5, A_6, A_7 are equal to 1). Hence, the genetic operators transformed the clients of C_2 from *HFC* into *thin* clients. For this reason a new server farm SF_3 for user class C_2 has been created.

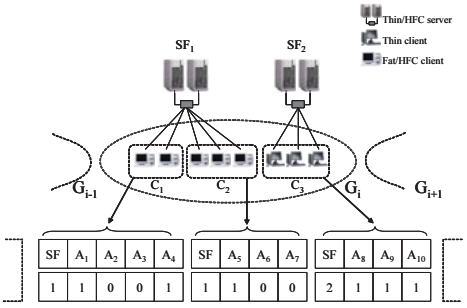

Fig. 1. Chromosome representation of the system.

Fixed applications represent applications that have been assigned to clients or to server farms. In this case the mapping of the chromosome is different. Only applications declared as *Free* by the designer are coded into the chromosome since only they represent the *variables* of the problem (Figure 3). Such a kind of requirements change the permitted ranges for chromosome integers. If a class have fixed application to server farms, then the clients cannot become FAT, hence the integer value SF must start from 1, see class C_3 in Figure 3. Consequent, users of class C_3 will be always connected to a server farm. On the other hand, if some applications have been fixed by the designer on the client, then the

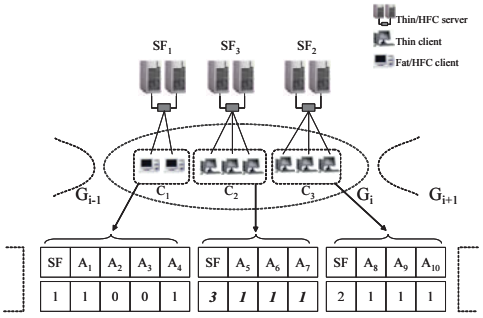

Fig. 2. Mutations and cross-over operator applied to the chromosome shown in Fig. 1.

client cannot become *thin* (see class C_1 in Figure 3). It is important to note that *each* configuration of the chromosome (obtained by mutating the integers in their proper ranges) correspond to a systems which completely fulfills the requirements expressed by the designer, but with different costs.

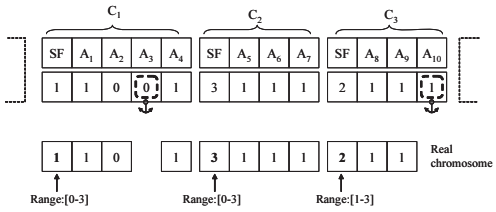

Fig. 3. Chromosome representation for fixed applications indicated by dashed boxes.

Let us define the remaining elements of the genetic algorithm. The selection function is the *roulette wheel*, and the genetic operators we used are *simple cross-over*[3] and *binary mutation* [13]. Initial populations have been created by randomly initializing the chromosomes of individuals. The termination criterion chosen is the maximum number of generations. The evaluation function is the cost of the current system. Since the fitting function is a cost, the proposed problem is a minimization of the fitting of the individuals.

4 Experimental Results

Empirical verifications were performed in order to evaluate the effectiveness of our cost minimization approach. Analyses are based on a database of commercial

[3] The simple cross-over operator exchanges the genes of two chromosomes satisfying the admitted ranges of integers.

components and related cost data. Management costs of physical components have been estimated by means of information provider benchmarks ([8], [9]). Cost effectiveness is evaluated by comparing the optimization's output with the output that can be obtained by applying the following *professional guidelines*:

1. Thin clients and HFCs are adopted whenever possible to minimize the cost of clients.
2. Server farms are implemented whenever possible and designed by selecting the smallest server that can support applications, to reduce hardware acquisition costs.
3. Users classes in the same group are centralized on a single server/server farm, to reduce management costs.

The case study we propose considers two groups. The first group (Data-Entry workers) contains 4 classes which use Word, Excel, IE6 and Outlook. Data-Entry workers are associated with Fat/Thin/NFC clients and all applications can be allocated on the remote server or on the client. The second group (Power-User workers) contains 4 classes and is assigned to a Fat/HFC client architecture. Users use Word, Excel, IE6 and Outlook, which can be allocated on the server or on the client, and Matlab is allocated on the client.

Results are reported in Table 2. It is straightforward that if all the applications are free (the Data-Entry group), the thin philosophy gives better results. Note anyway that, over 2000 users, the genetic optimization found a minimum cost system by partitioning users in two server farms (last row of the first group). This contradicts the #3 rule of professional guidelines.

In the power user group, due to the fixed application on the client side (Matlab) and to its computational-intensive nature, the best systems are composed by fat clients. The genetic optimization proves that it does not exist an intermediate hybrid solution that is less expensive. Again the results seem to contradict classical guidelines which tend to propose hybrid solutions.

Results highlight that the design of a complex system should always consider an optimization of the hardware infrastructure. Evolutionary techniques properly face this computationally intensive optimization allowing for considerable overall cost reductions. Experiments performed by using a tabu-search engine confirm the results obtained by the proposed genetic engine and showed that the parameters' tuning of the genetic optimizer is not critical. A suitable large initial population (50 individuals) and a moderately large number of optimization epochs (500) tends to produce acceptable solutions.

5 Conclusions

Professional guidelines can lead to sub-optimal architectural choices. A methodological approach and evolutionary optimization can improve solutions and they should be always considered during the design phases by allowing considerable cost reductions.

Table 2. Comparison between designing strategies.

Data-Entry Group
Freely allocable applications

	Optimization strategies				**Saving**	
Users	Guidelines	#SF	Optimized	#SF	Total	per User
12	31180 $	1	**31180** $	1	0 $	0 $
100	248900 $	1	**248900** $	1	0 $	0 $
500	1234790 $	1	**1234790** $	1	0 $	0 $
2000	4985010 $	1	**4973560** $	2	11450 $	5.72 $

Power-User Group
Freely allocable applications except Matlab

	Optimization strategies				**Saving**	
Users	Guidelines	#SF	Optimized	#SF	Total	per User
12	101928 $	1	**94704** $	0	7224 $	602.00 $
100	618560 $	1	**5919000** $	0	26660 $	266.60 $
500	3130270 $	1	**2959500** $	0	170770 $	341.54 $
1000	6675966 $	1	**5919000** $	0	756966 $	756.96 $

References

1. Ardagna, D., Francalanci, C., Piuri. V. Designing and Rightsizing the Information System Architecture. To appear on Information Systems Frontiers, Kluwer Academic Publishers.
2. Ardagna, D., Francalanci, C., Trubian, M. 2004. A Cost-oriented Approach for Inftrastructural Design. SAC 2004 Proceedings. Nicosia, Cyprus.
3. Ein-Dor, P. 1985. Groschs Law Revisited: CPU Power and the Cost of Computing. Management Communication of the ACM. 28(2), 142-150.
4. Gavish, B., Pirkul, H. 1986. Computer and Database Location in Distributed Computer Systems. IEEE Transaction on Computers. 35(7), 583-590.
5. Jain, H. K. 1987. A comprehensive model for the design of distributed computer systems. IEEE Transactions on software engineering. 13(10), 1092-1104.
6. Dewire D. T. 1997. Second-generation Client/Server Computing. McGraw Hill.
7. Menasce, D. A., Alameida, V. A. F. 2000. Scaling for E-business. Technologies, models, performance and capacity planning. Prentice-Hall.
8. Molta, D. 1999. Thin Client computers come of age. Network Computing. www.networkcomputing.com/1009/1009buyers1.html.
9. The Tolly Group. 1999. Total Cost of Application Ownership. www.tolly.com.
10. Microsoft. 2003. Windows Server 2003 Terminal Server Capacity and Scaling. www.microsoft.com/windowsserver2003/ techinfo/overview/tsscaling.mspx.
11. Sun 2003. Sun Ray Overview. White Paper. http://wwws.sun.com/sunray/whitepapers/
12. Papadimitriou, C., Steiglitz K. 1982. Combinatorial Optimization. Prentice Hall.
13. Koza, J., R. 1998. Genetic programming Encyclopedia of Computer Science and Technology volume 39, James G. Williams and Allen Kent 29–43
14. Houck, C., Joines, J., Kay, M. 1996. A genetic algorithm for functionoptimization: A Matlab implementation, citeseer.nj.nec.com/houck96genetic.html